PASSED

732A38: Computer lab 2
Computational statistics

Martina Sandberg and Caroline Svahn

February 11, 2016

1 Optimizing a model parameter

File *mortality_rate.csv* contains information about mortaily rates of the fruit flies during a certain period.

1.1

Import this file to R and add one more variable LMR to the data which is the natural logarithm of Rate. Afterwards, divide the data into training and test sets.

1.2

Write your own function `myMSE` *that for given parameters* **lambda** *and list* **pars** *containing vectors X, Y, Xtest, Ytest fits a LOESS model with response Y and predictor X using* `loess()` *function with penalty* **lambda** *(parameter* `enp.target` *in* `loess()`*) and then predicts the model for Xtest. The function should compute the predictive MSE, print it and return as a result.*

The `loess()` function is used for fitting local polynomial regression models. The parameter λ is used to control the span of the neighborhood used for smoothing. The predictive MSE is calculated as:

$$MSE = \frac{1}{n}\sum_{j=1}^{n}(\hat{Y}_j - Y_j)^2 \tag{1}$$

We use the variable "LMR" as Y vector, so the variables in the training set is Y and the variables in the test set is Ytest. In the same way we use the variable "Day" as X vector, so the variables in the training set is X and the variables in the test set is Xtest. In the function `myMSE` we fit a LOESS model with response Y and predictor X and then predicts the model for Xtest. The output from the prediction ($\hat{Y}$) is then used to compute the MSE according to formula (1) where Y corresponds to the vector Ytest.

1.3

Use a simple approach: use function `myMSE`*, training and test sets and lambda values: 0.1,0.2,. . . ,40, to estimate the predictive MSE values. Create a plot of the MSE values versus lambda and comment on which lambda value is optimal. How many evaluations of* `myMSE` *were required to find this value?*

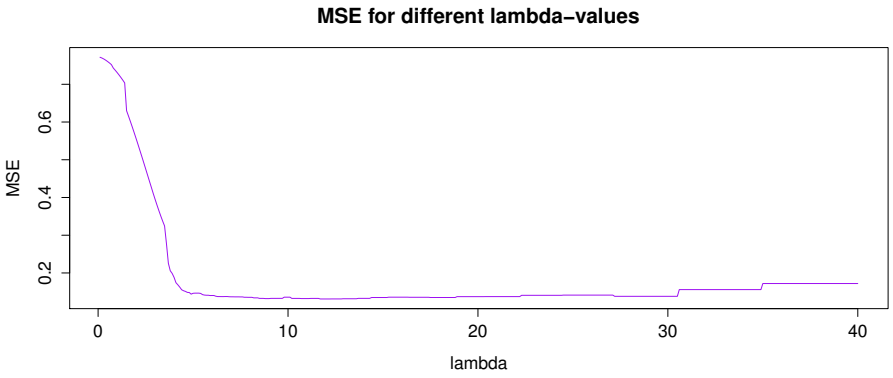

Figure 1: MSE for different lambda values.

In figure 1 the MSE values for different lambda values are plotted. The plot shows a steep fall in MSE for $\lambda < 5$. The MSE then stabilizes somewhat and appears to reach a minimum value around $\lambda = 11$. With R, the minimum MSE can be determined as 0.1310 at $\lambda = 11.7, 11.8, 11.9, 12.0, 12.1, 12.2$. The first minimum is at $\lambda = 11.7$ and is therefore the optimal. The number of evaluations required to find the minimum is equal to the number of iterations, since we evaluate every MSE for the range $0.1 \leq \lambda \leq 40$. Thus, the number of function evaluations required is $40/0.1 = 400$.

1.4

Use `optimize()` *function for the same purpose, specify range for search [0.1, 40] and the accuracy 0.01. Have the function managed to find the optimal MSE value? How many myMSE function evaluations were required? Compare to step 3.*

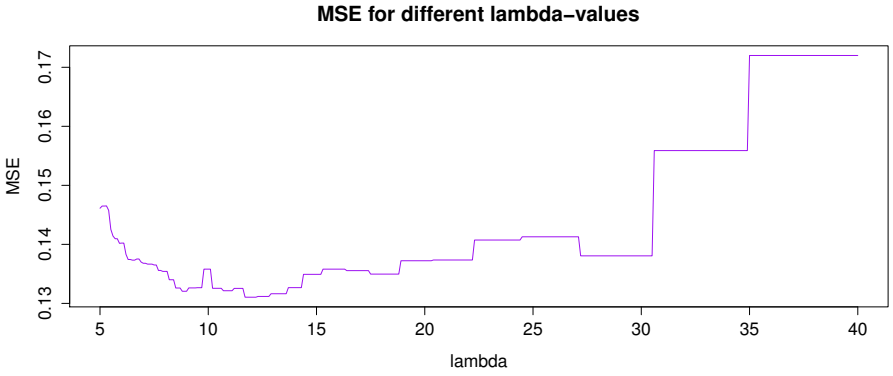

Figure 2: MSE for different lambda values.

The function `optimize()` search for the minimum (or maximum, depending on settings) in a set interval. When setting the accuracy to 0.01, the function stops when an improvement of no more than of 0.01 in the target function to optimize is obtained. Therefore, the entire interval is not necessarily searched.

When using the function `optimize()` we get MSE 0.1321 at lambda value 10.69, although this MSE continues at lambda values 10.7, 10.8, 10.9, 11.0 and 11.1. This function does not manage to find the optimal MSE but is still pretty close. It hard to see in figure 1 exactly how the curve looks like so in figure 2 we zoom in at the lambda values 5-40. Here we can see that this function is stuck on a local minimum. To find this MSE value required 18 evaluations of `myMSE`, which is far less than in step 3.

1.5

*Use `optim()` function and **BFGS** method with starting point lambda=35 to find the optimal lambda value. How many `myMSE` function evaluations were required? Compare the results you obtained with the results from step 4 and make conclusions.*

When using the function `optim()` we get MSE 0.1720 at lambda value 35, although this MSE continues at lambda values 35-40. This function does not manage to find the optimal MSE and is also far from it. To find this MSE value required 3 evaluations of `myMSE`, which is far less than in steps 3-4.This behaviour is because of the fact that the MSE function drops around $\lambda = 35$ (since it is now backtracking to find the minimum) and then flattens out completely. Since the derivative is then zero, the `optim()` function concludes a minimum is found and stops.

Comparing the three approaches to find the minimum, we find that evaluating all possible values garantees the global minimum to be found, but requires a lot of iterations and with a lot of data, this might take too much time. The `optimimize()` function does not search the entire span and here, requires sub-

stantially less evaluations to find an approximal minumum. The `optim()` function, however, requires only three evaluations but the MSE found is not even close to the actual minimum. Since the brute force approach requires so many evaluations and the derivative solution is so dependent on the initial point, the best option is the `optimize()` function, since it does not require that many iterations but still finds an MSE close to the global minimum.

2 Maximizing likelihood

File *data.RData* contains a sample from normal distribution with some parameters μ, σ.

2.1

Load the data to R environment.

2.2

Write down the log-likelihood function for 100 observations and derive maximum likelihood estimators for μ, σ analytically by setting partial derivatives to zero. Use formulas derived to obtain parameter estimates for the data loaded.

$$L(\mu, \sigma^2 | x_1, \ldots, x_{100}) = P(x_1, \ldots, x_{100} | \mu, \sigma^2) = f(x_1, \ldots, x_{100} | \mu, \sigma^2) \quad (2)$$

$$L(\mu, \sigma^2 | x_1, \ldots, x_{100}) = \frac{1}{\sqrt{\sigma^2 2\pi}^{100}} exp \left\{ -\frac{1}{2\sigma^2} \sum_{i=1}^{100} (x_i - \mu)^2 \right\} \quad (3)$$

$$\left.\begin{aligned} l(\mu,\sigma^2|x_1,\ldots,x_{100})) &= ln\left(\frac{1}{\sqrt{\sigma^2 2\pi}^{100}} exp\left\{-\frac{1}{2\sigma^2}\sum_{i=1}^{100}(x_i-\mu)^2\right\}\right) \\ &= ln\left(\frac{1}{\sqrt{\sigma^2 2\pi}^{100}}\right) + ln\left(exp\left\{-\frac{1}{2\sigma^2}\sum_{i=1}^{100}(x_i-\mu)^2\right\}\right) \\ &= ln(1) - ln(\sqrt{\sigma^2 2\pi}^{100}) - \frac{1}{2\sigma^2}\sum_{i=1}^{100}(x_i-\mu)^2 \\ &= -50ln(\sigma^2 2\pi) - \frac{1}{2\sigma^2}\sum_{i=1}^{100}(x_i-\mu)^2 \\ &= -50ln(2\pi) - 50ln(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{100}(x_i-\mu)^2 \end{aligned}\right\} \tag{4}$$

$$\left.\begin{aligned} \frac{\delta}{\delta\mu} l(\mu,\sigma^2|x_1,\ldots,x_{100})) = 0 \\ \frac{\delta}{\delta\sigma^2} l(\mu,\sigma^2|x_1,\ldots,x_{100})) = 0 \end{aligned}\right\} \tag{5}$$

$$\left.\begin{aligned} \frac{\delta}{\delta\mu}\left(-50ln(2\pi) - 50ln(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{100}(x_i-\mu)^2\right) \\ = \frac{1}{\sigma^2}\sum_{i=1}^{100}(x_i-\mu) = \frac{1}{\sigma^2}\left(\sum_{i=1}^{100}x_i - 100\mu\right) \end{aligned}\right\} \tag{6}$$

$$\sum_{i=1}^{100} x_i - 100\mu = 0 \tag{7}$$

$$\mu = \frac{1}{100}\sum_{i=1}^{100} x_i \tag{8}$$

$$\left.\begin{aligned} \frac{\delta}{\delta\sigma}\left(-50ln(2\pi) - 50ln(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{100}(x_i-\mu)^2\right) \\ = -\frac{100}{\sigma} + \frac{1}{\sigma^3}\sum_{i=1}^{100}(x_i-\mu)^2 \end{aligned}\right\} \tag{9}$$

$$\sigma^2 = \frac{1}{100}\sum_{i=1}^{100}(x_i - \mu)^2 \tag{10}$$

The likelihood is defined as (2), so the likelihood for 100 observations of the normal distribution is (3). The log likelihood $ln(L(\mu, \sigma^2|x_1, \ldots, x_{100})) = l(\mu, \sigma^2|x_1, \ldots, x_{100}))$ are derived as (4). We can derive maximum likelihood estimators for μ and σ analytically by setting partial derivatives to zero (5). The partial derivative of μ becomes (6) which is zero when (7) which gives us the maximum likelihood estimator (8). The partial derivative of σ becomes (9), we put this to zero and solve for σ, here we assume that $\sigma \neq 0$, (10). The maximum likelihood estimators for μ and σ is therefore (11), which is the sample mean and the unadjusted sample variance. When using these formulas on the data provided we get $\mu = 1.2755$ and $\sigma = 2.0060$.

$$\left.\begin{aligned} \hat{\mu} &= \frac{1}{100}\sum_{i=1}^{100} x_i = 1.2755 \\ \hat{\sigma}^2 &= \frac{1}{100}\sum_{i=1}^{100}(x_i - \hat{\mu})^2 = 4.0239 \end{aligned}\right\} \tag{11}$$

2.3

Now you are assumed to derive maximum likelihood estimates numerically. Why it is a bad idea to maximize likelihood rather than maximizing log-likelihood?

The reason why we maximize log-likelihood rather than maximizing likelihood is because the log-likelihood is often easier to work with. The logarithm of a function have the same maximum value as the function itself and can therefore be used instead of the function. As we can see in part 2 we have a product of individual likelihood functions and the logarithm of a product is a sum of individual logarithms which is much easier to work with when we derivate. We also have an exponent which gets much easier to work with when we take log of it.

2.4

Optimize the minus log-likelihood function with initial parameters $\mu = 0, \sigma = 1$. Try both Conjugate Gradient method and BFGS algorithm with gradient specified and without.

Here we use the function `optim`. First we use the methods BFGS and CG without specifying gradients (we call them BFGS and CG) and then when do the

same but specify the gradients to be the partial derivatives of the minus log-likelihood function (we call them BFGS-GRAD and CG-GRAD). That is for μ (12) and for σ (13).

$$\frac{1}{\sigma^2}\sum_{i=1}^{100}(x_i - \mu) \tag{12}$$

$$-\frac{100}{\sigma} + \frac{1}{\sigma^3}\sum_{i=1}^{100}(x_i - \mu)^2 \tag{13}$$

The main difference between the conjugate gradient method and the BFGS is that BFGS tends to be more "costly" since it uses the Hessian, which then needs to be found.

2.5

Did algorithms converge in all cases? What were the optimal values of parameters and how many function and gradient evaluations it required for algorithms to converge? Which settings would you recommend?

	function	gradient
BFGS	37	15
CG	180	33
BFGS_GRAD	38	15
CG_GRAD	53	17

Table 1: Number of evaluations required for the algorithms to converge.

The algorithms converged in all cases.
In table 1 the number of function and gradient evaluations required for the algorithms to converge is shown. The conjugate gradient function requires both more function evaluations and more gradient evaluations, with and without specified gradient. Quite a few less evaluations are needed in general when specifying the gradient function. When we look at the time for the algorithms to execute (using `system.time()` in R), BFGS, BFGS-GRAD and CG-GRAD takes 0.001 seconds and CG takes 0.004 seconds. CG is the function that performs the worst in this case.

The optimal values of the parameters were the same in all cases and was for $\mu = 1.2755$ and for $\sigma = 2.0060$. The expectation is, however, that a better result would be found when specifying the gradient, since, when one does not provide a gradient function, `optim()` uses a finite approximation of the gradient. The lack of difference is probably due to the fact that the data is merely 2 dimensional. The gradient is therefore rather simple and the approximation of

the gradient is quite a precise one.

It appears both algorithms - with and without specified gradient - finds values for μ and σ which are close to the expected values, and they all converge, therefore all methods works in theory. The BFGS does however find the values after less evaluations. Hence, the BFGS without specified gradient is the best option here.

Appendix

```
## Assignment 1

# 1.
data <- read.csv("~/Dropbox/LiU/CS/Labs/LAB/mortality_rate.csv", sep=";", dec="
LMR <- log(data[["Rate"]])
data <- cbind(data,LMR)

n=dim(data)[1]
set.seed(123456)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]

# 2.
myMSE <- function(lambda,pars){
  m <- loess(pars$Y~pars$X,enp.target=lambda)
  Yhat <- predict(m, newdata = pars$Xtest)
  n <- length(Yhat)
  MSE <- (1/n)*sum( (Yhat-pars$Ytest)^2 )
  print(MSE)
  return(MSE)
}

# 3.
li <- list(X=train[["Day"]],Y=train[["LMR"]],Xtest=test[["Day"]],Ytest=test[["L

MSE <- numeric()
for( i in seq(from = 0.1, to = 40, by = 0.1)){
  MSE <- append(MSE,myMSE(i,li))
}

plot(y=MSE,x=seq(from = 0.1, to = 40, by = 0.1),type="l",xlab="lambda",col="pu

# Zoom
MSE <- numeric()
for( i in seq(from = 5, to = 40, by = 0.1)){
  MSE <- append(MSE,myMSE(i,li))
}
```

```
plot(y=MSE,x=seq(from = 5, to = 40, by = 0.1),type="l",xlab="lambda",col="purpl
###

min(MSE)
#seq(from = 0.1, to = 40, by = 0.1)[match(min(MSE),MSE)]
seq(from = 0.1, to = 40, by = 0.1)[which(MSE %in% min(MSE))]
# MSE = 0.131047
# lambda = 11.7 11.8 11.9 12.0 12.1 12.2
# evaluations = 400


# 4.

opti <- optimize(myMSE,interval=c(0.1, 40), tol = 0.01,pars=li)
seq(from = 0.1, to = 40, by = 0.1)[which(MSE %in% opti$objective)]
# MSE = 0.1321441
# lambda = 10.7 10.8 10.9 11.0 11.1
# evaluations = 18


# 5.

op <- optim(par=35,fn=myMSE,method="BFGS",pars=li)
#seq(from = 0.1, to = 40, by = 0.1)[match(op$value,MSE)]
seq(from = 0.1, to = 40, by = 0.1)[which(MSE %in% op$value)]
# MSE = 0.1719996
# lambda = 35-40
# evaluations = 3


## Assignment 2

# 1.
load("data.RData")


# 2.

mu <- mean(data)
# mu = 1.275528

sigma2 <- sum((data-mu)^2)/100
# sigma2 = 4.023942
# sigma = 2.005977
# unadjusted variance
# var(data) = 4.064587
# sum((data-mu)^2)/99 = 4.064587


# 3.
```

```
# 4.

nL <- function(pars,x){
  mu <- pars[1]
  sigma2 <- pars[2]^2
  n <- length(x)
  logl <- (n/2)*log(2*pi)+(n/2)*log(sigma2)+(1/(2*sigma2))*sum((x-mu)^2)
  return(logl)
}

# WITHOUT SPECIFIED GRADIENT
# If gr = NULL, a finite-difference approximation will be used.

# Method "BFGS" is a quasi-Newton method
opt1 <- optim(c(0,1),nL,method="BFGS",x=data)
matrix(opt$par,2,1)
# mu = 1.275528
# sigma2 = 2.005977^2 = 4.023944



# Method "CG" is a conjugate gradients method
opt2 <- optim(c(0,1),nL,method="CG",x=data)

# WITH SPECIFIED GRADIENT

gradient <- function(pars,x){
  mu <- pars[1]
  sigma <- pars[2]
  n <- length(x)
  grad1 <- -(1/sigma^2)*sum(x-mu)
  grad2 <- (n/sigma)-( (1/sigma^3)*sum((x-mu)^2) )
  return(c(grad1,grad2))
}


# Method "BFGS" is a quasi-Newton method
opt3 <- optim(c(0,1),nL,method="BFGS",gr=gradient,x=data)



# Method "CG" is a conjugate gradients method
opt4 <- optim(c(0,1),nL,method="CG",gr=gradient,x=data)


# BFGS
opt1$counts
# CG
opt2$counts
# BFGS grad
```

```
opt3$counts
# CG grad
opt4$counts


system.time(optim(c(0,1),nL,method="BFGS",x=data))
user  system elapsed
0.001   0.000   0.001
> system.time(optim(c(0,1),nL,method="CG",x=data))
user  system elapsed
0.004   0.000   0.004
> system.time(optim(c(0,1),nL,method="BFGS",gr=gradient,x=data))
user  system elapsed
0.001   0.000   0.001
> system.time(optim(c(0,1),nL,method="CG",gr=gradient,x=data))
user  system elapsed
0.001   0.000   0.001
```